Nastran Analysis Interface Module (AIM)

Ryan Durscher
AFRL/RQVC

0.1 Introduction

0.1.1 Nastran AIM Overview

A module in the Computational Aircraft Prototype Syntheses (CAPS) has been developed to interact (primarily through input files) with the finite element structural solver Nastran [1].

Details on the use of units are outlined in AIM Units.

An outline of the AIM's inputs, outputs and attributes are provided in AIM Inputs and AIM Outputs and AIM attributes, respectively.

Details of the AIM's automated data transfer capabilities are outlined in AIM Data Transfer

0.1.2 Examples

Example problems using the Nastran AIM may be found at Nastran AIM Examples .

- Single Load Case Example
- Multiple Load/Boundary Case Example
- Modal Analysis Example Case
- Optimization Example Case
- Composite Wing Example
- Composite Wing Optimization Example

0.1.3 Clearance Statement

This software has been cleared for public release on 05 Nov 2020, case number 88ABW-2020-3462.

0.2 AIM attributes

The following list of attributes are required for the Nastran AIM inside the geometry input.

- **capsDiscipline** This attribute is a requirement if doing aeroelastic analysis within Nastran. capsDiscipline allows the AIM to determine which bodies are meant for structural analysis and which are used for aerodynamics. Options are: Structure and Aerodynamic (case insensitive).
- **capsGroup** This is a name assigned to any geometric body to denote a property. This body could be a solid, surface, face, wire, edge or node. Recall that a string in ESP starts with a $. For example, attribute `capsGroup $Wing`.
- **capsLoad** This is a name assigned to any geometric body where a load is applied. This attribute was separated from the `capsGroup` attribute to allow the user to define a local area to apply a load on without adding multiple `capsGroup` attributes. Recall that a string in ESP starts with a $. For example, attribute `capsLoad $force`.

- **capsConstraint** This is a name assigned to any geometric body where a constraint/boundary condition is applied. This attribute was separated from the `capsGroup` attribute to allow the user to define a local area to apply a boundary condition without adding multiple `capsGroup` attributes. Recall that a string in ESP starts with a $. For example, attribute `capsConstraint $fixed`.

- **capsIgnore** It is possible that there is a geometric body (or entity) that you do not want the Nastran AIM to pay attention to when creating a finite element model. The capsIgnore attribute allows a body (or entity) to be in the geometry and ignored by the AIM. For example, because of limitations in OpenCASCADE a situation where two edges are overlapping may occur; capsIgnore allows the user to only pay attention to one of the overlapping edges.

- **capsConnect** This is a name assigned to any geometric body where the user wishes to create "fictitious" connections such as springs, dampers, and/or rigid body connections to. The user must manually specify the connection between two `capsConnect` entities using the "Connect" tuple (see AIM Inputs). Recall that a string in ESP starts with a $. For example, attribute `capsConnect $springStart`.

- **capsConnectLink** Similar to `capsConnect`, this is a name assigned to any geometric body where the user wishes to create "fictitious" connections to. A connection is automatically made if a `capsConnectLink` matches a `capsConnect` group. Again further specifics of the connection are input using the "Connect" tuple (see AIM Inputs). Recall that a string in ESP starts with a $. For example, attribute `capsConnect↩ Link $springEnd`.

- **capsResponse** This is a name assigned to any geometric body that will be used to define design sensitivity responses for optimization. Specific information for the responses are input using the "Design_Response" tuple (see AIM Inputs). Recall that a string in ESP starts with a $. For examples, attribute `capsResponse $displacementNode`.

- **capsBound** This is used to mark surfaces on the structural grid in which data transfer with an external solver will take place. See AIM Data Transfer for additional details.

Internal Aeroelastic Analysis

- **capsSpline** This is used to mark FACEs/EDGEs/NODEs on the structural grid in which splines will be created between the structural and aero-loads.

- **capsReferenceArea** [Optional: Default 1.0] Reference area to use when doing aeroelastic analysis. This attribute may exist on any aerodynamic cross-section.

- **capsReferenceChord** [Optional: Default 1.0] Reference chord to use when doing aeroelastic analysis. This attribute may exist on any aerodynamic cross-section.

- **capsReferenceSpan** [Optional: Default 1.0] Reference span to use when doing aeroelastic analysis. This attribute may exist on any aerodynamic cross-section.

0.3 AIM Units

A unit system may be optionally specified during AIM instance initiation. If a unit system is provided, all AIM input values which have associated units must be specified as well. If no unit system is used, AIM inputs, which otherwise would require units, will be assumed unit consistent. A unit system may be specified via a JSON string dictionary for example (using pyCAPS):

```
nastran = capsProblem.analysis.create(aim="nastranAIM", unitSys = {"mass": "kg", "length": "inch", "time":
      "hour", "temperature" : "Kelvin"})
```

0.3.1 JSON String Dictionary

The key arguments of the dictionary are described in the following:

- **mass = "None"**
  Mass units - e.g. "kilogram", "k", "slug", ...

- **length = "None"**
  Length units - e.g. "meter", "m", "inch", "in", "mile", ...

- **time = "None"**
  Time units - e.g. "s", "second", "hour", "day", "year", ...

- **temperature = "None"**
  Temperature units - e.g. "C", "F", "K", "R", ...

0.4 AIM Inputs

The following list outlines the Nastran inputs along with their default value available through the AIM interface. Unless noted these values will be not be linked to any parent AIMs with variables of the same name.

- **Proj_Name = "nastran_CAPS"**
  This corresponds to the project name used for file naming.

- **Tess_Params = [0.025, 0.001, 15.0]**
  Body tessellation parameters used when creating a boundary element model. Tess_Params[0] and Tess↩
  _Params[1] get scaled by the bounding box of the body. (From the EGADS manual) A set of 3 parameters that drive the EDGE discretization and the FACE triangulation. The first is the maximum length of an EDGE segment or triangle side (in physical space). A zero is flag that allows for any length. The second is a curvature-based value that looks locally at the deviation between the centroid of the discrete object and the underlying geometry. Any deviation larger than the input value will cause the tessellation to be enhanced in those regions. The third is the maximum interior dihedral angle (in degrees) between triangle facets (or Edge segment tangents for a WIREBODY tessellation), note that a zero ignores this phase

- **Edge_Point_Min = 2**
  Minimum number of points on an edge including end points to use when creating a surface mesh (min 2).

- **Edge_Point_Max = 50**
  Maximum number of points on an edge including end points to use when creating a surface mesh (min 2).

- **Quad_Mesh = False**
  Create a quadratic mesh on four edge faces when creating the boundary element model.

- **Echo = "NONE"**
  Sets the the Bulk Data ECHO statement.
  May be: "SORT", "UNSORT", "NONE"

- **Nastran = NULL**
  List of strings to specify NASTRAN statements
  The NASTRAN statement is used to specify values for certain Executive System operational parameters. These parameters are also called system cells. The NASTRAN statement is used for exceptional circumstances
  and is therefore not needed in most runs. The NASTRAN statement may also be specified in the
  runtime configuration (RC) files at the system, user, and job level as described in the .

- **Property = NULL**
  Property tuple used to input property information for the model, see FEA Property for additional details.

- **Material = NULL**
  Material tuple used to input material information for the model, see FEA Material for additional details.

- **Constraint = NULL**
  Constraint tuple used to input constraint information for the model, see FEA Constraint for additional details.

- **Load = NULL**
  Load tuple used to input load information for the model, see FEA Load for additional details.

- **Analysis = NULL**
  Analysis tuple used to input analysis/case information for the model, see FEA Analysis for additional details.

- **Analysis_Type = "Modal"**
  Type of analysis to generate files for, options include "Modal", "Static", "AeroelasticTrim", "AeroelasticFlutter", and "Optimization". Note: "Aeroelastic" and "StaticOpt" are still supported and refer to "AeroelasticTrim" and "Optimization".

- **File_Format = "Small"**
  Formatting type for the bulk file. Options: "Small", "Large", "Free".

- **Mesh_File_Format = "Small"**
  Formatting type for the mesh file. Options: "Small", "Large", "Free".

- **Design_Variable = NULL**
  The design variable tuple is used to input design variable information for the model optimization, see FEA Design Variables for additional details.

- **Design_Variable_Relation = NULL**
  The design variable relation tuple is used to input design variable relation information for the model optimization, see FEA DesignVariableRelation for additional details.

- **Design_Constraint = NULL**
  The design constraint tuple is used to input design constraint information for the model optimization, see FEA Design Constraints for additional details.

- **Design_Equation = NULL**
  The design equation tuple used to input information defining equations for use in design sensitivity, see FEA Design Equations for additional details.

- **Design_Table = NULL**
  The design table tuple used to input table of real constants used in equations, see FEA Table Constants for additional details.

- **Design_Response = NULL**
  The design response tuple used to input design sensitivity response information, see FEA Design Responses for additional details.

- **Design_Equation_Response = NULL**
  The design equation response tuple used to input design sensitivity equation response information, see FEA Design Equation Responses for additional details.

- **Design_Opt_Param = NULL**
  The design optimization parameter tuple used to input parameters used in design optimization, see FEA Design Optimization Parameters for additional details.

- **Objective_Min_Max = "Max"**
  Maximize or minimize the design objective during an optimization. Option: "Max" or "Min".

- **Objective_Response_Type = "Weight"**
  Object response type (see Nastran manual).

- **Mass_Increment = NULL**
  FEA Mass Increments (see Nastran manual).
- **VLM_Surface = NULL**
  Vortex lattice method tuple input, see Vortex Lattice Surface for additional details.
- **VLM_Control = NULL**
  Vortex lattice method control surface tuple input, see Vortex Lattice Control Surface for additional details.
- **Support = NULL**
  Support tuple used to input support information for the model, see FEA Support for additional details.
- **Connect = NULL**
  Connect tuple used to define connection to be made in the, see FEA Connection for additional details.
- **Parameter = NULL**
  Parameter tuple used to define PARAM entries. Note, entries are output exactly as inputed, that is, if the PARAM entry requires an integer entry the user must input an integer!
- **Model_Parameter = NULL**
  Model_Parameter tuple used to define MDLPRM entries. Note, entries are output exactly as inputed, that is, if the MDLPRM entry requires an integer entry the user must input an integer!
- **Aero_Reference = NULL**
  A JSON dictionary used to define aerodynamic reference parameters. see FEA Aerodynamic References for additional details
- **VLM_Camber_Twist = True**
  Apply camber & twist to VLM sections. Option: True or False.
- **Visualize_Flutter = False**
  Determines if flutter cards are written for visualization. Option: True or False.
- **Mesh_Morph = False**
  Project previous surface mesh onto new geometry.
- **Mesh = NULL**
  A Mesh link.

0.5 AIM Outputs

The following list outlines the Nastran outputs available through the AIM interface.

- **EigenValue** = List of Eigen-Values (λ) after a modal solve.
- **EigenRadian** = List of Eigen-Values in terms of radians ($\omega = \sqrt{\lambda}$) after a modal solve.
- **EigenFrequency** = List of Eigen-Values in terms of frequencies ($f = \frac{\omega}{2\pi}$) after a modal solve.
- **EigenGeneralMass** = List of generalized masses for the Eigen-Values.
- **EigenGeneralStiffness** = List of generalized stiffness for the Eigen-Values.
- **Objective** = Final objective value for a design optimization case.
- **ObjectiveHistory** = List of objective value for the history of a design optimization case.
- **Mass** = Total mass of the model.
- **CG** = Center of gravity of the model.
- **Ixx** = Moment of inertia

- **Iyy** = Moment of inertia
- **Izz** = Moment of inertia
- **Ixy** = Moment of inertia
- **Izy** = Moment of inertia
- **Iyz** = Moment of inertia
- **I_Vector** = Moment of inertia vector

$$\vec{I} = \begin{bmatrix} I_{xx} & I_{yy} & I_{zz} & I_{xy} & I_{xz} & I_{yz} \end{bmatrix}$$

- **I_Lower** = Moment of inertia lower triangular tensor

$$\vec{I}_{lower} = \begin{bmatrix} I_{xx} & -I_{xy} & I_{yy} & -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix},$$

- **I_Upper** = Moment of inertia upper triangular tensor

$$\vec{I}_{upper} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} & I_{yy} & -I_{yz} & I_{zz} \end{bmatrix},$$

- **I_Tensor** = Moment of inertia tensor

$$\bar{\bar{I}} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$$

- **MassProp** = JSON String Mass Properties
- **MassPropLink** = Mass Properties Mass properties that can be linked to analysis input MassPropLink

0.6 AIM Data Transfer

The Nastran AIM has the ability to transfer displacements and eigenvectors from the AIM and pressure distributions to the AIM using the conservative and interpolative data transfer schemes in CAPS.

0.6.1 Data transfer from Nastran (FieldOut)

- **"Displacement"**
  Retrieves nodal displacements from the $*$.f06 file.

- **"EigenVector_#"**
  Retrieves modal eigen-vectors from the $*$.f06 file, where "#" should be replaced by the corresponding mode number for the eigen-vector (eg. EigenVector_3 would correspond to the third mode, while EigenVector_6 would be the sixth mode).

0.6.2 Data transfer to Nastran (FieldIn)

- **"Pressure"**
  Writes appropriate load cards using the provided pressure distribution.

0.7 FEA Material

Structure for the material tuple = ("Material Name", "Value"). "Material Name" defines the reference name for the material being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.7.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"density": 7850, "youngModulus": 120000.0, "poissonRatio": 0.5, "materialType": "isotropic"}) the following keywords (= default values) may be used:

- **materialType = "Isotropic"**
  Material property type. Options: Isotropic, Anisothotropic, Orthotropic, or Anisotropic.

- **youngModulus = 0.0**
  Also known as the elastic modulus, defines the relationship between stress and strain. Default if 'shear↩ Modulus' and 'poissonRatio' != 0, youngModulus = 2∗(1+poissonRatio)∗shearModulus

- **shearModulus = 0.0**
  Also known as the modulus of rigidity, is defined as the ratio of shear stress to the shear strain. Default if 'youngModulus' and 'poissonRatio' != 0, shearModulus = youngModulus/(2∗(1+poissonRatio))

- **poissonRatio = 0.0**
  The fraction of expansion divided by the fraction of compression. Default if 'youngModulus' and 'shear↩ Modulus' != 0, poissonRatio = (2∗youngModulus/shearModulus) - 1

- **density = 0.0**
  Density of the material.

- **thermalExpCoeff = 0.0**
  Thermal expansion coefficient of the material.

- **thermalExpCoeffLateral = 0.0**
  Thermal expansion coefficient of the material.

- **temperatureRef = 0.0**
  Reference temperature for material properties.

- **dampingCoeff = 0.0**
  Damping coefficient for the material.

- **yieldAllow = 0.0**
  Yield strength/allowable for the material.

- **tensionAllow = 0.0**
  Tension allowable for the material.

- **tensionAllowLateral = 0.0**
  Lateral tension allowable for the material.

- **compressAllow = 0.0**
  Compression allowable for the material.

- **compressAllowLateral = 0.0**
  Lateral compression allowable for the material.

- **shearAllow = 0.0**
  Shear allowable for the material.

- **allowType = 0**
  This flag defines if the above allowables `compressAllow` etc. are defined in terms of stress (0) or strain (1). The default is stress (0).

- **youngModulusLateral = 0.0**
  Elastic modulus in lateral direction for an orthotropic material

- **shearModulusTrans1Z = 0.0**
  Transverse shear modulus in the 1-Z plane for an orthotropic material

- **shearModulusTrans2Z = 0.0**
  Transverse shear modulus in the 2-Z plane for an orthotropic material

- **kappa = 0.0**
  Thermal conductivity for an isotropic solid

- **K = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]**
  Thermal conductivity for an anisotropic solid (KXX, KXY, KXZ, KYY, KYZ, KZZ)

- **specificHeat = 0.0**
  Specific heat constant pressure (per unit mass) for an isotropic solid

- **Gij = (no default)**
  List of Gij material properties (e.g. [G11, G12, G13, G22, fG23, G33]). Length must be 6.

- **honeycombCellSize = NULL**
  Honeycomb sandwich core cell size. Required if material defines the core of a honeycomb sandwich and dimpling stability index is desired

- **honeycombYoungModulus = NULL**
  Honeycomb sandwich core Young's modulus used for stability index analysis

- **honeycombShearModulus = NULL**
  Honeycomb sandwich core shear modulus used for stability index analysis

- **fractureAngle = NULL**
  Fracture angle for uniaxial transverse compression in degrees. Used in the NASA LaRC02 failure theory only

- **interlaminarShearAllow = NULL**
  Allowable inter-laminar shear stress of the composite laminate bonding material

- **fiberYoungModulus = NULL**
  Modulus of elasticity of fiber

- **fiberPoissonRatio = NULL**
  Poisson's ratio of fiber

- **meanStressFactor = NULL**
  Mean stress magnification factor

- **transTensionSlope = NULL**
  Failure envelop slope parameter for transverse tension

- **transCompressionSlope = NULL**
  Failure envelop slope parameter for transverse compression

- **compositeFailureTheory = NULL**
  Composite failure theory (string value)

- **interlaminarNormalStressAllow = NULL**
  Allowable inter-laminar normal stress of the composite laminate bonding material (allowable interlaminar normal stress)

- **youngModulusThick = NULL**
  Modulus of elasticity in thickness direction, also defined as the matrix direction or 3-direction

- **poissonRatio23 = 0.0**
  Poisson's ratio (for uniaxial loading in 2-direction)

- **poissonRatio31 = 0.0**
  Poisson's ratio (for uniaxial loading in 3-direction)

- **youngModulusFactor = NULL**
  Longitudinal modulus of elasticity reduction scale factor for nonlinear composite Progressive Ply Failure Analysis (PPFA)

- **youngModulusLateralFactor = NULL**
  Lateral modulus of elasticity reduction scale factor for nonlinear composite Progressive Ply Failure Analysis (PPFA)

- **shearModulusFactor = NULL**
  In-plane shear modulus reduction scale factor for nonlinear composite Progressive Ply Failure Analysis (PPFA)

- **shearModulusTrans1ZFactor = NULL**
  Transverse shear modulus reduction scale factor in 1-Z plane for nonlinear composite Progressive Ply Failure Analysis (PPFA)

- **shearModulusTrans2ZFactor = NULL**
  Transverse shear modulus reduction scale factor in 2-Z plane for nonlinear composite Progressive Ply Failure Analysis (PPFA)

0.7.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined material lookup table. NOT YET IMPLEMENTED!!!!

0.8 FEA Property

Structure for the property tuple = ("Property Name", "Value"). "Property Name" defines the reference `capsGroup` for the property being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.8.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"shearMembraneRatio": 0.83, "bendingInertiaRatio": 1.0, "membraneThickness": 0.2, "propertyType": "Shell"}) the following keywords (= default values) may be used:

- **propertyType = No Default value**
  Type of property to apply to a given capsGroup `Name`. Options: ConcentratedMass, Rod, Bar, Shear, Shell, Composite, and Solid

- **material = "Material Name" (FEA Material)**
  "Material Name" from FEA Material to use for property.
  Defaults to material index if 1 if only one material has been provided.

- **crossSecArea = 0.0**
  Cross sectional area.

- **torsionalConst = 0.0**
  Torsional constant.

- **torsionalStressReCoeff = 0.0**
  Torsional stress recovery coefficient.

- **massPerLength = 0.0**
  Non-structural mass per unit length.

- **zAxisInertia = 0.0**
  Section moment of inertia about the element z-axis.

- **yAxisInertia = 0.0**
  Section moment of inertia about the element y-axis.

- **yCoords[4] = [0.0, 0.0, 0.0, 0.0]**
  Element y-coordinates, in the bar cross-section, of four points at which to recover stresses

- **zCoords[4] = [0.0, 0.0, 0.0, 0.0]**
  Element z-coordinates, in the bar cross-section, of four points at which to recover stresses

- **areaShearFactors[2] = [0.0, 0.0]**
  Area factors for shear.

- **crossProductInertia = 0.0**
  Section cross-product of inertia.

- **crossSecType = NULL**
  Cross-section type. Must be one of following character variables: BAR, BOX, BOX1, CHAN, CHAN1, CHAN2, CROSS, H, HAT, HEXA, I, I1, ROD, T, T1, T2, TUBE, or Z.

- **crossSecDimension = [0,0,0,....]**
  Cross-sectional dimensions (length of array is dependent on the "crossSecType"). Max supported length array is 10!

- **membraneThickness = 0.0**
  Membrane thickness.

- **bendingInertiaRatio = 1.0**
  Ratio of actual bending moment inertia to the bending inertia of a solid plate of thickness "membrane↩ Thickness"

- **shearMembraneRatio = 5.0/6.0**
  Ratio shear thickness to membrane thickness.

- **materialBending = "Material Name" (FEA Material)**
  "Material Name" from FEA Material to use for property bending. If no material is given and "bendingInertia↩ Ratio" is greater than 0, the material name provided in "material" is used.

- **materialShear = "Material Name" (FEA Material)**
  "Material Name" from FEA Material to use for property shear. If no material is given and "shearMembrane↩ Ratio" is greater than 0, the material name provided in "material" is used.

- **massPerArea = 0.0**
  Non-structural mass per unit area.

- **zOffsetRel = 0.0**
  Relative offset from the surface of grid points to the element reference plane as a percentage of the thickness. zOffSet = thickness∗zOffsetRel/100

- **compositeMaterial = "no default"**
  List of "Material Name"s, ["Material Name -1", "Material Name -2", ...], from FEA Material to use for composites.

- **shearBondAllowable = 0.0**
  Allowable interlaminar shear stress.

- **symmetricLaminate = False**
  Symmetric lamination option. True- SYM only half the plies are specified, for odd number plies 1/2 thickness of center ply is specified with the first ply being the bottom ply in the stack, default (False) all plies specified.

- **compositeFailureTheory = "(no default)"**
  Composite failure theory. Options: "HILL", "HOFF", "TSAI", and "STRN"

- **compositeThickness = (no default)**
  List of composite thickness for each layer (e.g. [1.2, 4.0, 3.0]). If the length of this list doesn't match the length of the "compositeMaterial" list, the list is either truncated [>length("compositeMaterial")] or expanded [<length("compositeMaterial")] in which case the last thickness provided is repeated.

- **compositeOrientation = (no default)**
  List of composite orientations (angle relative element material axis) for each layer (eg. [5.0, 10.0, 30.0]). If the length of this list doesn't match the length of the "compositeMaterial" list, the list is either truncated [>length("compositeMaterial")] or expanded [<length("compositeMaterial")] in which case the last orientation provided is repeated.

- **mass = 0.0**
  Mass value.

- **massOffset = [0.0, 0.0, 0.0]**
  Offset distance from the grid point to the center of gravity for a concentrated mass.

- **massInertia = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]**
  Mass moment of inertia measured at the mass center of gravity.

0.8.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined property lookup table. NOT YET IMPLEMENTED!!!!

0.9 FEA Constraint

Structure for the constraint tuple = ("Constraint Name", "Value"). "Constraint Name" defines the reference name for the constraint being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.9.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"groupName": "plateEdge", "dofConstraint": 123456}) the following keywords (= default values) may be used:

- **constraintType = "ZeroDisplacement"**
  Type of constraint. Options: "Displacement", "ZeroDisplacement".

- **dofConstraint = 0**
  Component numbers / degrees of freedom that will be constrained (123 - zero translation in all three directions).

- **gridDisplacement = 0.0**
  Value of displacement for components defined in "dofConstraint".

0.9.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined constraint lookup table. NOT YET IMPLEMENTED!!!!

0.10 FEA Support

Structure for the support tuple = ("Support Name", "Value"). "Support Name" defines the reference name for the support being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.10.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"groupName": "plateEdge", "dofSupport": 123456}) the following keywords (= default values) may be used:

- **groupName = "(no default)"**
  Single or list of `capsConstraint` names on which to apply the support (e.g. "Name1" or ["Name1","↩ Name2",...]. If not provided, the constraint tuple name will be used.

- **dofSupport = 0**
  Component numbers / degrees of freedom that will be supported (123 - zero translation in all three directions).

0.10.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined support lookup table. NOT YET IMPLEMENTED!!!!

0.11 FEA Connection

Structure for the connection tuple = ("Connection Name", "Value"). "Connection Name" defines the reference name to the capsConnect being specified and denotes the "source" node for the connection. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.11.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"dofDependent": 1, "propertyType": "RigidBody"}) the following keywords (= default values) may be used:

- **connectionType = RigidBody**
  Type of connection to apply to a given capsConnect pair defined by "Connection Name" and the "groupName". Options: Mass (scalar), Spring (scalar), Damper (scalar), RigidBody, RigidBodyInterpolate.

- **dofDependent = 0**
  Component numbers / degrees of freedom of the dependent end of rigid body connections (ex. 123 - translation in all three directions).

- **componentNumberStart = 0**
  Component numbers / degrees of freedom of the starting point of the connection for mass, spring, and damper elements (scalar) (0 <= Integer <= 6).

- **componentNumberEnd= 0**
  Component numbers / degrees of freedom of the ending point of the connection for mass, spring, damper elements (scalar), and rigid body interpolative connection (0 <= Integer <= 6).

- **stiffnessConst = 0.0**
  Stiffness constant of a spring element (scalar).

- **dampingConst = 0.0**
  Damping coefficient/constant of a spring or damping element (scalar).

- **stressCoeff = 0.0**
  Stress coefficient of a spring element (scalar).

- **mass = 0.0**
  Mass of a mass element (scalar).

- **weighting = 1**
  Weighting factor for a rigid body interpolative connections.

- **groupName = "(no default)"**
  Single or list of `capsConnect` names on which to connect the nodes found with the tuple name ("↩ Connection Name") to. (e.g. "Name1" or ["Name1","Name2",...].

0.11.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined connection lookup table. NOT YET IMPLEMENTED!!!!

0.12 FEA Load

Structure for the load tuple = ("Load Name", "Value"). "Load Name" defines the reference name for the load being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.12.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"groupName": "plate", "loadType": "Pressure", "pressureForce": 2000000.0}) the following keywords (= default values) may be used:

- **loadType = "(no default)"**
  Type of load. Options: "GridForce", "GridMoment", "Rotational", "Thermal", "Pressure", "PressureDistribute", "PressureExternal", "ThermalExternal", "Gravity".

- **groupName = "(no default)"**
  Single or list of `capsLoad` names on which to apply the load (e.g. "Name1" or ["Name1","Name2",...]. If not provided, the load tuple name will be used.

- **loadScaleFactor = 1.0**
  Scale factor to use when combining loads.

- **forceScaleFactor = 0.0**
  Overall scale factor for the force for a "GridForce" load.

- **directionVector = [0.0, 0.0, 0.0]**
  X-, y-, and z- components of the force vector for a "GridForce", "GridMoment", or "Gravity" load.

- **momentScaleFactor = 0.0**
  Overall scale factor for the moment for a "GridMoment" load.

- **gravityAcceleration = 0.0**
  Acceleration value for a "Gravity" load.

- **pressureForce = 0.0**
  Uniform pressure force for a "Pressure" load.

- **pressureDistributeForce = [0.0, 0.0, 0.0, 0.0]**
  Distributed pressure force for a "PressureDistribute" load. The four values correspond to the 4 (quadrilateral elements) or 3 (triangle elements) node locations.

- **angularVelScaleFactor = 0.0**
  An overall scale factor for the angular velocity in revolutions per unit time for a "Rotational" load.

- **angularAccScaleFactor = 0.0**
  An overall scale factor for the angular acceleration in revolutions per unit time squared for a "Rotational" load.

- **coordinateSystem = "(no default)"**
  Name of coordinate system in which defined force components are in reference to. If no value is provided the global system is assumed.

- **temperature = 0.0**
  Temperature at a given node for a "Temperature" load.

- **temperatureDefault = 0.0**
  Default temperature at a node not explicitly being used for a "Temperature" load.

0.12.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined load lookup table. NOT YET IMPLEMENTED!!!!

0.13 FEA Analysis

Structure for the analysis tuple = ('Analysis Name', 'Value'). 'Analysis Name' defines the reference name for the analysis being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.13.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"numDesiredEigenvalue": 10, "eigenNormalization": "MASS", "numEstEigenvalue": 1, "extractionMethod": "GIV", "frequencyRange": [0, 10000]}) the following keywords (= default values) may be used:

- **analysisType = "Modal"**
  Type of load. Options: "Modal", "Static", "AeroelasticTrim", "AeroelasticFlutter" Note: "AeroelasticStatic" is still supported but refers to "AeroelasticTrim" Note: "Optimization" and "StaticOpt" are not valid - Optimization is initialized by the Analysis_Type AIM Input

- **analysisLoad = "(no default)"**
  Single or list of "Load Name"s defined in FEA Load in which to use for the analysis (e.g. "Name1" or ["↩ Name1","Name2",...].

- **analysisConstraint = "(no default)"**
  Single or list of "Constraint Name"s defined in FEA Constraint in which to use for the analysis (e.g. "Name1" or ["Name1","Name2",...].

- **analysisSupport = "(no default)"**
  Single or list of "Support Name"s defined in FEA Support in which to use for the analysis (e.g. "Name1" or ["Name1","Name2",...].

- **analysisDesignConstraint = "(no default)"**
  Single or list of "Design Constraint Name"s defined in FEA Design Constraints in which to use for the analysis (e.g. "Name1" or ["Name1","Name2",...].

- **analysisCaseMassIncrement = "(no default)"**
  Single or list of "Concentrated Mass Property Name"s defined in in which to use for the analysis (e.g. "Name1" or ["Name1","Name2",...].

- **extractionMethod = "(no default)"**
  Extraction method for modal analysis.

- **frequencyRange = [0.0, 0.0]**
  Frequency range of interest for modal analysis.
  Note: specifying inf (math.inf or np.inf in Python) results in a blank entry in the input card

- **numEstEigenvalue = 0**
  Number of estimated eigenvalues for modal analysis.

- **numDesiredEigenvalue = 0**
  Number of desired eigenvalues for modal analysis.
  A value <= 0 will result in a blank card entry

- **eigenNormalization = "(no default)"**
  Method of eigenvector renormalization. Options: "POINT", "MAX", "MASS"

- **gridNormalization = 0**
  Grid point to be used in normalizing eigenvector to 1.0 when using eigenNormalization = "POINT"

- **componentNormalization = 0**
  Degree of freedom about "gridNormalization" to be used in normalizing eigenvector to 1.0 when using eigen↩
  Normalization = "POINT"

- **lanczosMode = 2**
  Mode refers to the Lanczos mode type to be used in the solution. In mode 3 the mass matrix, Maa,must be nonsingular whereas in mode 2 the matrix K aa - sigma∗Maa must be nonsingular

- **lanczosType = "(no default)"**
  Lanczos matrix type. Options: DPB, DGB.

- **machNumber = 0.0 or [0.0, ..., 0.0]**
  Mach number used in trim analysis OR Mach numbers used in flutter analysis..

- **dynamicPressure = 0.0**
  Dynamic pressure used in trim analysis.

- **density = 0.0**
  Density used in trim analysis to determine true velocity, or flutter analysis.

- **aeroSymmetryXY = "(no default)"**
  Aerodynamic symmetry about the XY Plane. Options: SYM, ANTISYM, ASYM. Aerodynamic symmetry about the XY Plane. Options: SYM, ANTISYM, ASYM. SYMMETRIC Indicates that a half span aerodynamic model is moving in a symmetric manner with respect to the XY plane. ANTISYMMETRIC Indicates that a half span aerodynamic model is moving in an antisymmetric manner with respect to the XY plane. ASYMMETRIC Indicates that a full aerodynamic model is provided.

- **aeroSymmetryXZ = "(no default)"**
  Aerodynamic symmetry about the XZ Plane. Options: SYM, ANTISYM, ASYM. SYMMETRIC Indicates that a half span aerodynamic model is moving in a symmetric manner with respect to the XZ plane. ANTISYMMETRIC Indicates that a half span aerodynamic model is moving in an antisymmetric manner with respect to the XZ plane. ASYMMETRIC Indicates that a full aerodynamic model is provided.

- **rigidVariable = ["no default"]**
  List of rigid body motions to be used as trim variables during a trim analysis. Nastran valid labels are: ANGLEA, SIDES, ROLL, PITCH, YAW, URDD1, URDD2, URDD3, URDD4, URDD5, URDD6

- **rigidConstraint = ["no default"]**
  List of rigid body motions to be used as trim constraint variables during a trim analysis. Nastran valid labels are: ANGLEA, SIDES, ROLL, PITCH, YAW, URDD1, URDD2, URDD3, URDD4, URDD5, URDD6

- **magRigidConstraint = [0.0 , 0.0, ...]**
  List of magnitudes of trim constraint variables. If none and 'rigidConstraint'(s) are specified then 0.0 is assumed for each rigid constraint.

- **controlConstraint = ["no default"]**
  List of controls surfaces to be used as trim constraint variables during a trim analysis.

- **magControlConstraint = [0.0 , 0.0, ...]**
  List of magnitudes of trim control surface constraint variables. If none and 'controlConstraint'(s) are specified then 0.0 is assumed for each control surface constraint.

- **reducedFreq = [0.1, ..., 20.0], No Default Values are defined.**
  Reduced Frequencies to be used in Flutter Analysis. Up to 8 values can be defined.

- **flutterVel = [0.1, ..., 20.0]**
  Velocities to be used in Flutter Analysis. If no values are provided the following relation is used

  v = sqrt(2∗dynamicPressure/density) dv = (v∗2 - v/2) / 20;

  flutterVel[0] = v/10 flutterVel[i] = v/2 + i∗dv; where i = 1....21 flutterVel[22] = v∗10;

- **visualFlutter = False**
  Turn on flutter visualization f06 output.

- **analysisResponse = "(no default)"**
  Single or list of "DesignResponse Name"s defined in FEA Design Responses to use for the analysis response spanning sets (e.g. "Name1" or ["Name1","Name2",...].

0.13.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined analysis lookup table. NOT YET IMPLEMENTED!!!!

0.14 FEA Design Variables

Structure for the design variable tuple = ("DesignVariable Name", "Value"). "DesignVariable Name" defines the reference name for the design variable being specified. This string will be used in the FEA input directly. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary). In Nastran the DesignVariable Name will be the LABEL used in the DESVAR input. For this reason the user should keep the length of this input to a minimum number of characters, ideally 7 or less.

- `DESVAR ID LABEL XINIT XLB XUB DELXV DDVAL`

0.14.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"initialValue": 5.0, "upperBound": 10.0}) the following keywords (= default values) may be used:

- **initialValue = 0.0**
  Initial value for the design variable.

- **lowerBound = 0.0**
  Lower bound for the design variable.

- **upperBound = 0.0**
  Upper bound for the design variable.

- **maxDelta = 0.5**
  Change fraction limit for the design variable.

- **discreteValue = 0.0**
  List of discrete values to use for the design variable (e.g. [0.0,1.0,1.5,3.0].

- **independentVariable = "(no default)"**
  Single or list of "DesignVariable Name"s (that is the Tuple name) used to create/designate a dependent design variable.

  – independentValue = variableWeight[1] + variableWeight[2] $*$ SUM{independentVariableWeight[i] $*$ independentVariable[i]}

- **independentVariableWeight = 1.0 or [1.0, 1.0, ...]**
  Single or list of weighting constants with respect to the variables set for "independentVariable". If the length of this list doesn't match the length of the "independentVariable" list, the list is either truncated [$>$length("independentVariable")] or expanded [$<$length("independentVariable")] in which case the **last weight is repeated**.

- **variableWeight = [1.0, 1.0]**
  Weighting constants for a dependent variable - used if "independentVariable"(s) have been provided.

0.15 FEA DesignVariableRelation

Structure for the design variable tuple = ("DesignVariableRelation Name", "Value"). "DesignVariableRelation Name" defines the reference name for the design variable being specified. This string will be used in the FEA input directly. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

0.15.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"componentType": "Property", "componentName": "plate", "fieldName": "TM", "variableName": "MyDesVar"}) the following keywords (= default values) may be used:

- **componentType = "Property"**
  The type of component for this design variable relation. Options: "Material", "Property", "Element".

- **componentName = "(no default)"**
  Single or list of FEA Property(ies), or FEA Material name(s) linked to the design variable relation (e.g. "↩ Name1" or ["Name1","Name2",...].
  - For `componentType` Property a FEA Property name (or names) is given.
  - For `componentType` Material a FEA Material name (or names) is given.
  - For `componentType` Element a `capsGroup` Name (or names) is given.

- **variableName = "(no default)"**
  Single or list of names of design variables linked to this relation

- **fieldName = "(no default)"**
  Fieldname of variable relation (e.g. "E" for Young's Modulus). Design Variable Relations can be defined as three types based on the `variableType` value. These are Material, Property, or Element. This means that an aspect of a material, property, or element input can change in the optimization problem. This input specifies what aspect of the Material, Property, or Element is changing.

  1. **Material Types** Selected based on the material type (see FEA Material, materialType) referenced in the `componentName` above.
     - **`MAT1,`** `materialType = "Isotropic"`
       - "E", "G", "NU", "RHO", "A"
     - **`MAT2,`** `materialType = "Anisothotropic"`
       - "G11", "G12", "G13", "G22", "G23", "G33", "RHO", "A1", "A2", "A3"
     - **`MAT8,`** `materialType = "Orthotropic"`
       - "E1", "E2", "NU12", "G12", "G1Z", "G2Z", "RHO", "A1", "A2"
     - **`MAT9,`** `materialType = "Anisotropic"`
       - "G11", "G12", "G13", "G14", "G15", "G16"
       - "G22", "G23", "G24", "G25", "G26"
       - "G33", "G34", "G35", "G36"
       - "G44", "G45", "G46"
       - "G55", "G56", "G66"
       - "RHO", "A1", "A2", "A3", "A4", "A5", "A6"
  2. **Property Types** (see FEA Property)
     - **`PROD`** `propertyType = "Rod"`
       - "A", "J"
     - **`PBAR`** `propertyType = "Bar"`
       - "A", "I1", "I2", "J"
     - **`PSHELL`** `propertyType = "Shell"`
       - "T"
     - **`PCOMP`** `propertyType = "Composite"`
       - "T1", "THETA1", "T2", "THETA2", ... "Ti", "THETAi"
     - **`PSOLID`** `propertyType = "Solid"`
       - not supported

3. **Element Types**
   - **CTRIA3, CQUAD4** `propertyType = "Shell"`
     * "ZOFFS"

- **fieldPosition = 0**
  This input is ignored if not defined. The user may use this field instead of the `fieldName` input defined above to relate design variables and property, material, or elements. This requires knowledge of Nastran bulk data input format for material, property, and element input cards.

- **constantCoeff = 0.0**
  Constant term of relation.

- **linearCoeff = 1.0**
  Single or list of coefficients of linear relation. Must be same length as `variableName`.

0.16 FEA Design Constraints

Structure for the design constraint tuple = ('DesignConstraint Name', 'Value'). 'DesignConstraint Name' defines the reference name for the design constraint being specified. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

0.16.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"groupName": "plate", "upperBound": 10.0}) the following keywords (= default values) may be used:

- **groupName = "(no default)"**
  Single or list of `capsGroup` name(s) to the design variable (e.g. "Name1" or ["Name1","Name2",...].The property (see FEA Property) also assigned to the same `capsGroup` will be automatically related to this constraint entry.

- **constraintType = "Property"**
  The type of design constraint. Options: "Property", "Flutter"

- **lowerBound = 0.0**
  Lower bound for the design constraint.

- **upperBound = 0.0**
  Upper bound for the design constraint.

- **responseType = "(no default)"**
  Response type options for DRESP1 Entry (see Nastran manual).
  - Implemented Options
    1. `STRESS`, for `propertyType = "Rod" or "Shell"` (see FEA Property)
    2. `CFAILURE`, for `propertyType = "Composite"` (see FEA Property)

- **fieldName = "(no default)"**
  For constraints, this field is only used currently when applying constraints to composites. This field is used to identify the specific lamina in a stacking sequence that a constraint is being applied too. Note if the user has design variables for both THEATA1 and T1 it is likely that only a single constraint on the first lamina is required. For this reason, the user can simply enter LAMINA1 in addition to the possible entries defined in the FEA Design Variables section. Additionally, the `fieldPosition` integer entry below can be used. In this case `"LAMINA1" = 1`.

  – -# **Property Types** (see FEA Property)

    * **PCOMP** `propertyType = "Composite"`
      · "T1", "THETA1", "T2", "THETA2", ... "Ti", "THETAi"
      · "LAMINA1", "LAMINA2", ... "LAMINAi"

- **fieldPosition = 0**
  This input is ignored if not defined. The user may use this field instead of the `fieldName` input defined above to identify a specific lamina in a composite stacking sequence where a constraint is applied. Please read the `fieldName` information above for more information.

0.16.2 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"groupName": "plate", "upperBound": 10.0}) the following keywords (= default values) may be used:

- **groupName = "(no default)"**
  Single or list of `capsGroup` name(s) to the design variable (e.g. "Name1" or ["Name1","Name2",...].The property (see FEA Property) also assigned to the same `capsGroup` will be automatically related to this constraint entry.

0.17 FEA Optimization Control

Structure for the optimization control dictionary = 'Value'. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

- **fullyStressedDesign = 0**
  Number of iterations with fully stressed design.

- **mathProgramming = 30**
  Number of iterations for math programming methods.

- **maxIter = 30**
  Maximum number of optimization iterations.

- **constraintRetention = 1.5**
  Constraint retention factor. Will be at least 1.5 times the number of design variables

- **eps = 1.0**
  Constraint retention parameter in which all constraints having a value greater than "eps" will be considered active.

- **moveLimit = 1.0**
  Move limit bound.

0.18 FEA Mass Increments

Structure for the mass increment tuple = (‘MassIncrement Name’, ‘Value’). 'MassIncrement Name' defines the reference name for the mass increment being specified. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

0.19 FEA Design Equations

Structure for the design equation tuple = ("DesignEquation Name", ["Value1", ... , "ValueN"]). "DesignEquation Name" defines the reference name for the design equation being specified. This string will be used in the FEA input directly. The values "Value1", ... , "ValueN" are a list of strings containing the equation defintions. (see Section List of equation strings).

0.19.1 List of equation strings

Each design equation tuple value is a list of strings containing the equation definitions (eg. ["dispsum3(s1,s2,s3)=sum(s1,s2,s3)"]

0.20 FEA Table Constants

Structure for the table constant tuple = ("TableConstant Name", "Value"). "TableConstant Name" defines the reference name for the table constant being specified. This string will be used in the FEA input directly. The "Value" is the value of the table constant. In Nastran the TableConstant Name will be the LABLi used in the DTABLE input. For this reason the user should keep the length of this input to a minimum number of characters, ideally 7 or less.

- `DTABLE LABL1 VALU1 LABL2 VALU2 LABL3 VALU3 -etc-`

0.21 FEA Design Responses

Structure for the design response tuple = ("DesignResponse Name", "Value"). "DesignResponse Name" defines the reference name for the design response being specified. This string will be used in the FEA input directly. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary). In Nastran the DesignResponse Name will be the LABEL used in the DRESP1 input. For this reason the user should keep the length of this input to a minimum number of characters, ideally 7 or less.

- `DRESP1 ID LABEL RTYPE PTYPE REGION ATTA ATTB ATT1 ATT2 -etc-`

0.21.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"responseType": "DISP", groupName": "plate", "component": 3}) the following keywords (= default values) may be used:

- **responseType**
  Type of design sensitivity response. For options, see NASTRAN User Guide DRESP1 Design Sensitivity Response Attributes table.

- **component = "(no default)"**
  Component flag.

- **attb = "(no default)"**
  ATTB Inputs.

- **groupName = "(no default)"**
  Defines the reference `capsGroup` for the node being specified for the response.

0.22 FEA Design Equation Responses

Structure for the design equation response tuple = ("DesignEquationResponse Name", "Value"). "DesignEquation↩ Response Name" defines the reference name for the design equation response being specified. This string will be used in the FEA input directly. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary). In Nastran the DesignEquationResponse Name will be the LABEL used in the DRESP2 input. For this reason the user should keep the length of this input to a minimum number of characters, ideally 7 or less.

- `DRESP2 ID LABEL EQID REGION ...`

0.22.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"equation": "EQ1", "constant": ["PI", "YM", "L"]}) the following keywords (= default values) may be used:

- **equation**
  The name of the equation referenced by this equation response.

- **variable = "(no default)"**
  Single or list of names of design variable equation parameters.

- **constant = "(no default)"**
  Single or list of names of table constant equation parameters.

- **response = "(no default)"**
  Single or list of names of design response equation parameters.

- **equationResponse = "(no default)"**
  Single or list of names of design equation response equation parameters.

0.23 FEA Design Optimization Parameters

Structure for the design optimization parameter tuple = ("DesignOptParam Name", "Value"). "DesignOptParam Name" defines the reference name for the design optimization parameter being specified. This string will be used in the FEA input directly. The "Value" is the value of the design optimization parameter. In Nastran the DesignOpt↩ Param Name will be the PARAMi used in the DOPTPRM input. For this reason the user should keep the length of this input to a minimum number of characters, ideally 7 or less.

- `DOPTPRM PARAM1 VAL1 PARAM2 VAL2 PARAM3 VAL3 -etc-`

0.24 FEA Aerodynamic References

Tuple of the aerodynamic reference input (see Section JSON String Dictionary).

0.24.1 JSON String Dictionary

The following keywords (= default values) may be used:

0.25 Vortex Lattice Surface

Structure for the Vortex Lattice Surface tuple = ("Name of Surface", "Value"). "Name of surface defines the name of the surface in which the data should be applied. The "Value" can either be a JSON String dictionary (see Section \ref jsonStringVLMSurface) or a single string keyword string (see Section \ref keyStringVLMSurface). @section jsonStringVLMSurface JSON String Dictionary If "Value" is a JSON string dictionary (eg. "Value" = {"numChord": 5, "spaceChord": 1.0, "numSpan": 10, "spaceSpan": 0.5}) the following keywords (= default values) may be used:

- **groupName = "(no default)"**
  Single or list of *capsGroup* names used to define the surface (e.g. "Name1" or ["Name1","Name2",...]. If no groupName variable is provided an attempted will be made to use the tuple name instead;

- **numChord = 10**
  The number of chordwise horseshoe vortices placed on the surface. Note: The chordwise count may be overridden using the *vlmNumChord* BODY attribute on a section.

- **spaceChord = 0.0**
  The chordwise vortex spacing parameter.

- **numSpanTotal = 0**
  Total number of spanwise horseshoe vortices placed on the surface. The vorticies are 'evenly' distributed across sections to minimize jumps in spacings. numpSpanPerSection must be zero if this is set.
  Note: The local spanwise count may be overridden using the *vlmNumSpan* BODY attribute on a section.

- **numSpanPerSection = 0**
  The number of spanwise horseshoe vortices placed on each section the surface. The total number of spanwise vorticies are (numSection-1)∗numSpanPerSection. The vorticies are 'evenly' distributed across sections to minimize jumps in spacings. numSpanTotal must be zero if this is set.
  Note: The local spanwise count may be overridden using the *vlmNumSpan* BODY attribute on a section.

- **spaceSpan = 0.0**
  The spanwise vortex spacing parameter.
  The local spacing may be overridden using the vlmSspace BODY attribute on a section.

- **sortVec = [0.0, 0.0, 0.0]**
  Vector for sorting airfoil sections.
  By default, section normals are used for sorting.
  However, for LINE sections, a sorting direction is necessary if sorting is desired.

- **interpSpline = NULL**
  A list of interpolating spline parameters, i.e.
  interpSpline = [{"splineName" : ["ribs", "spars"], "dz" : 0.1, method : "IPS"}, {"splineName" : "skin_top", "usage" : "FORCE", "method" : "TPS", "spanLimited": True}]

  Available options for each spline interpolation are:
  splineName : List of capsSpline attribute values for extracting grid IDs. Must be specified to connect VLM and structural mesh!
  dz : Linear attachment flexibility
  method : Method for the spline fit. IPS,TPS or FPS.
  usage : Spline usage flag to determine whether this spline applies to the force transformation, displacement transformation or both.
  nElem : The number of FE elements along the local spline x-axis if using the FPS option.
  mElem : The number of FE elements along the local spline y-axis if using the FPS option.
  spanLimited : Limit to use only spline points in-between airfoil sections.

0.25.1 Single Value String

If "Value" is a single string the following options maybe used:

- (NONE Currently)

0.26 Vortex Lattice Control Surface

Structure for the Vortex Lattice Control Surface tuple = ("Name of Control Surface", "Value"). "Name of control surface defines the name of the control surface in which the data should be applied. The "Value" must be a JSON String dictionary (see Section \ref jsonStringVLMSection). @section jsonStringVLMSection JSON String Dictionary If "Value" is a JSON string dictionary (e.g. "Value" = {"deflectionAngle": 10.0}) the following keywords (= default values) may be used: \iline 821 \iline 868 \iline 887 \iline 905 \iline 923 \iline 941 \iline 962 \iline 980 @section keyStringVLMControl Single Value String If "Value" is a single string, the following options maybe used:

- (NONE Currently)

0.27 Nastran AIM Examples

This section introduces the user to the Nastran AIM via examples. These examples are intended to introduce the user to nastran functionality. They make use of the information found in the AIM Inputs, AIM Outputs and AIM attributes sections.

0.27.1 Single Load Case Example

The first example is a simple three bar truss structure. This example is intended to demonstrate the use of all the attributes in addition to introducing the user to the Nastran AIM.

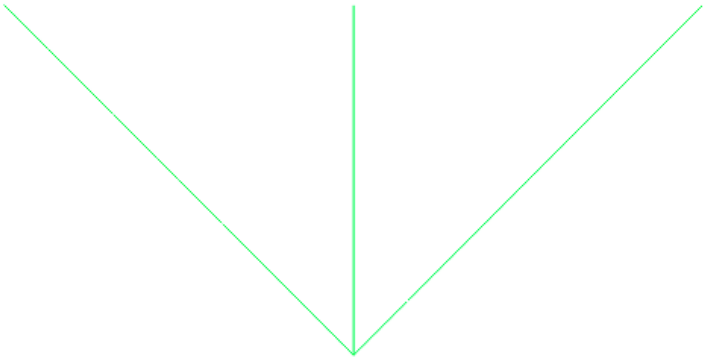

Figure 1 Three Bar Truss

The follow code details the process in a ∗.csm file that generates a three bar truss. Note to execute in serveESP a dictionary file must be included

- `serveESP feaThreeBar.csm`

The CSM script generates Bodies which are designed to be used by specific AIMs. The AIMs that the Body is designed for is communicated to the CAPS framework via the “capsAIM” string attribute. This is a semicolon-separated string with the list of AIM names. Thus, the CSM author can give a clear indication to which AIMs should use the Body. In this example, the list contains the structural finite element analysis tools that can analyze the body:

```
attribute capsAIM $nastranAIM;astrosAIM;mystranAIM;egadsTessAIM
```

Next we will define the design parameters to define the wing cross section and planform. Notice that the despmtr entries have a dimension input that must be defined for inputs with a length greater than one.

```
dimension   X   1 4 1
dimension   Y   1 4 1
dimension   Z   1 4 1

despmtr X   "-10; 0; 10;   0;"
despmtr Y   "  0; 0;  0; -10;"
despmtr Z   "  0; 0;  0;   0;"
```

Next the three bar truss is defined using the points defined in the `despmtr` entries. Notice that the middle edge is "drawn" twice. This is done because OpenCASCADE cannot perform boolean operations on non-manifold (not closed) wire bodies.

```
skbeg  X[1,1] Y[1,1] Z[1,1]
linseg X[1,4] Y[1,4] Z[1,4]
linseg X[1,2] Y[1,2] Z[1,2]
linseg X[1,4] Y[1,4] Z[1,4]
linseg X[1,3] Y[1,3] Z[1,3]
skend
```

In this section the edge elements are attributed with a `capsGroup` string `$bar1` etc. so information can be assigned to them. Notice the `capsIgnore` attribute assigned to one of the overlapping "middle" edges defined in the geometry above. The reason for this is discussed in the AIM attributes section.

```
select edge 1
attribute capsGroup $bar1
select edge 2
attribute capsGroup $bar2
select edge 3
attribute capsIgnore $multipleEdge
select edge 4
attribute capsGroup $bar3
```

Finally, the nodes are attributed. In this case, the three nodes across the top are given the same `capsConstraint` name. They could be assigned a different name allowing the user to define a different boundary condition at each location. The lower node is given a different `capsLoad` name so a load can be applied.

```
select node 1
attribute capsConstraint $boundary
select node 2
attribute capsLoad $force
select node 3
attribute capsConstraint $boundary
select node 4
attribute capsConstraint $boundary
```

The following input defines a pyCAPS input that can be used along with the above *.csm input to create a nastran input. First the pyCAPS and os module needs to be imported.

```
# Import pyCAPS class file
import pyCAPS

# Import os module
import os
import shutil
import argparse
```

Once the modules have been loaded the pyCAPS.Problem needs to be initiated while loading the *.csm file.

Though not shown in this example the user has access to the X, Y and Z despmtr inputs from this pyCAPS script.

The Nastran AIM is then loaded with:

```
# Load nastran aim
nastran = capsProblem.analysis.create(aim = "nastranAIM",
                                      name = "nastran")
```

After the AIM is loaded some of the inputs to the AIM are defined. A full list of options can be found in the AIM Inputs section. In this case the `Proj_Name` is entered. The project name becomes the Nastran input file names. Two are create `projectName.bdf` and `projectName.dat`. The `*.bdf` file contains the grid and connectivity information. The data file contains the case control and other bulk data inputs required by Nastran. The input format is selected as `Free` and large field format is used when the option is available. This is most likely in the GRID entries only. Additionally the analysis type selected is `Static`. The maximum and minimum points that can be placed along an edge is set to be two. This ensures that each edge shown in the figure will be represented by a single finite element bar.

```
nastran.input["Mesh"].link(egads.output["Surface_Mesh"])

nastran.input.Proj_Name = "threebar_nastran_Test"
nastran.input.File_Format = "Free"
nastran.input.Mesh_File_Format = "Large"
nastran.input.Analysis_Type = "Static"
```

Next the material inputs, property selection, constraints and loads are defined. First materials are defined.

```
madeupium    = {"materialType" : "isotropic",
                "youngModulus" : 1.0E7 ,
                "poissonRatio" : .33,
                "density"      : 0.1}

nastran.input.Material = {"Madeupium": madeupium}
```

Next these materials are used in the property definition. In this case two `bar` type properties are assigned to the edges. The outer bars have a property with a different area then the center bar. Note the relationship of `bar1` etc. between this pyCAPS input and the *,csm input previously shown.

```
rod  =   {"propertyType"      : "Rod",
          "material"          : "Madeupium",
          "crossSecArea"      : 1.0}

rod2  =   {"propertyType"     : "Rod",
           "material"         : "Madeupium",
           "crossSecArea"     : 2.0}

nastran.input.Property = {"bar1": rod,
                          "bar2": rod2,
                          "bar3": rod}
```

Next the three nodes with `capsConstraint` boundary are constrained in all six degrees of freedom.

```
constraint = {"groupName"          : ["boundary"],
              "dofConstraint"      : 123456}

nastran.input.Constraint = {"BoundaryCondition": constraint}
```

Finally a load is applied the the node with the `capsLoad` force.

```
load = {"groupName"         : "force",
        "loadType"          : "GridForce",
        "forceScaleFactor"  : 20000.0,
        "directionVector"   : [0.8, -0.6, 0.0]}

nastran.input.Load = {"appliedForce": load }
```

Finally an analysis case is defined that connects an analysis type to the load and constraint condition by name.

```
value = {"analysisType"         : "Static",
         "analysisConstraint"   : "BoundaryCondition",
         "analysisLoad"         : "appliedForce"}

capsProblem.analysis["nastran"].input.Analysis = {"SingleLoadCase": value }
```

Once all the inputs have been set, aimPreanalysis needs to be executed. During this operation all the necessary files to run Nastran are generated and placed in the analysis working directory (analysisDir)

```
nastran.preAnalysis()
```

An OS system call is then made from Python to execute Nastran.

```
print ("\n\nRunning Nastran......")

if args.noAnalysis == False:
    nastran.system("nastran old=no notify=no batch=no scr=yes sdirectory=./ " + nastran.input.Proj_Name +
      ".dat"); # Run Nastran via system call
else:
    nastran.system("test_bdf -e 0 " + nastran.input.Proj_Name +  ".dat"); # Run pyNastran test_bdf via
      system call
    # Copy old results if no analysis available
    shutil.copy2(os.path.join("..","analysisData","nastran",projectName+".f06"),
                 os.path.join(nastran.analysisDir,nastran.input.Proj_Name+".f06"))
    shutil.copy2(os.path.join("..","analysisData","nastran",projectName+".op2"),
                 os.path.join(nastran.analysisDir,nastran.input.Proj_Name+".op2"))

print ("Done running Nastran!")
```

A call to aimPostanalysis is then made to check to see if AVL executed successfully and the expected files were generated.

```
nastran.postAnalysis()
```

0.27.2 Multiple Load/Boundary Case Example

To create multiple load cases with different boundary conditions the pyCAPS input for constraints, and loads changes with respect to the Single Load Case Example. In addition an analysis section is added.

The constraint section may expand to allow multiple boundary conditions. In this way each load case can have a seperate boundary condition. If the input is left identical to the single load case example then the same boundary condition will be applied to each load case.

```
# Set constraints
conOne = {"groupName"         : ["boundary"],
          "dofConstraint"     : 123456}
conTwo = {"groupName"         : ["boundary"],
          "dofConstraint"     : 123}

nastran.input.Constraint = {"conOne": conOne, "conTwo": conTwo}
```

Notice that an empty `constraints` variable has been defined. Then a `tmp` tuple is created with the name `"conOne"` paired with the dictionary `constraint`. This tuple is appended to the empty `constraints` variable. The process is repeated for the second boundary condition. Then the AIM input `"Constraint"` is defined with the information.

Next the load input is expanded to contain multiple cases.

```
loadOne = {"groupName"         : "force",
        "loadType"          : "GridForce",
        "forceScaleFactor"  : 20000.0,
        "directionVector"   : [0.8, -0.6, 0.0]}

loadTwo = {"groupName"         : "force",
        "loadType"          : "GridForce",
        "forceScaleFactor"  : 20000.0,
```

```
        "directionVector"   : [-0.8, -0.6, 0.0]}

nastran.input.Load = {"loadOne": loadOne, "loadTwo": loadTwo}
```

The process is identical to the constraint input.

Finally, analysis cases are defined that connect an analysis type to a load and constraint condition by name.

```
caseOne = {"analysisType"         : "Static",
           "analysisConstraint"     : "conOne",
           "analysisLoad"         : "loadOne"}

caseTwo = {"analysisType"         : "Static",
           "analysisConstraint"     : "conTwo",
           "analysisLoad"         : "loadTwo"}

nastran.input.Analysis = {"caseOne": caseOne, "caseTwo": caseTwo}
```

Notice how the tuple names "conOne", "loadOne" and "analysisOne" are all tied together. The "analysisOne" string also becomes the case control LABEL for the load case in the Nastran input file.

To finish the pyCAPS input the process starting with the pre-analysis input is identical to the Single Load Case Example input.

0.27.3 Modal Analysis Example Case

To create input for a modal analysis a two simple changes are required to the Single Load Case Example input. The first change is to the AIM Inputs Analysis_Type. This input is the last input in the list below.

```
nastran.input["Mesh"].link(egads.output["Surface_Mesh"])
nastran.input.Proj_Name = "threebar_nastran_Test"
nastran.input.File_Format = "Free"
nastran.input.Mesh_File_Format = "Large"
nastran.input.Analysis_Type = "Static"
```

A description of each of these inputs can be found in Single Load Case Example.

The second change is replacing the load case information with a definition for the Analysis AIM input.

```
eigen = { "extractionMethod"     : "Lanczos",
          "frequencyRange"       : [0, 10000],
          "numEstEigenvalue"     : 1,
          "numDesiredEigenvalue" : 10,
          "eigenNormalization"   : "MASS"}

nastran.input.Analysis = {"EigenAnalysis": eigen}
nastran.input.Analysis_Type = "Modal"
```

This information defines the eigenvalue solver method and parameters and assigns it as an analysis case.

0.27.4 Optimization Example Case

This section creates a design model out of the single load case example. The first change is an update to the Analysis_Type AIM Input to StaticOpt.

```
nastran.input["Mesh"].link(egads.output["Surface_Mesh"])

nastran.input.Proj_Name = "threebar_nastran_Test"
nastran.input.File_Format = "Free"
nastran.input.Mesh_File_Format = "Large"
nastran.input.Analysis_Type = "StaticOpt"
```

The next update adds a material allowable to the material input yieldAllow. This is not a requirement for THIS optimization problem, but this input is referenced when design constraints are added later.

```
madeupium    = {"materialType" : "isotropic",
                "youngModulus" : 1.0E7 ,
                "poissonRatio" : .33,
                "density"      : 0.1,
```

```
                "yieldAllow"    : 5.6E7}

nastran.input.Material = {"Madeupium": madeupium}
```

The first large optimization input is the design variable definition section. For more information, the user is pointed to the FEA Design Variables section. In this section the area of each rod element in the three bar truss is defined as a separate design variable. Finally each of these variables are used to defined AIM Input `Design_Variable`

```
DesVar1    = {"initialValue" : rod["crossSecArea"],
              "lowerBound" : rod["crossSecArea"]*0.5,
              "upperBound" : rod["crossSecArea"]*1.5,
              "maxDelta"   : rod["crossSecArea"]*0.1}

DesVar2    = {"initialValue" : rod2["crossSecArea"],
              "lowerBound" : rod2["crossSecArea"]*0.5,
              "upperBound" : rod2["crossSecArea"]*1.5,
              "maxDelta"   : rod2["crossSecArea"]*0.1}

DesVar3    = {"initialValue" : rod["crossSecArea"],
              "lowerBound" : rod["crossSecArea"]*0.5,
              "upperBound" : rod["crossSecArea"]*1.5,
              "maxDelta"   : rod["crossSecArea"]*0.1}

nastran.input.Design_Variable = {"Bar1A": DesVar1,
                                 "Bar2A": DesVar2,
                                 "Bar3A": DesVar3}
```

The next unique section is the addition of design constraints. In this problem stress constraints in each rod element are added.

```
designConstraint1 = {"groupName" : "bar1",
                     "responseType" : "STRESS",
                     "lowerBound" : -madeupium["yieldAllow"],
                     "upperBound" :  madeupium["yieldAllow"]}

designConstraint2 = {"groupName" : "bar2",
                     "responseType" : "STRESS",
                     "lowerBound" : -madeupium["yieldAllow"],
                     "upperBound" :  madeupium["yieldAllow"]}

designConstraint3 = {"groupName" : "bar3",
                     "responseType" : "STRESS",
                     "lowerBound" : -madeupium["yieldAllow"],
                     "upperBound" :  madeupium["yieldAllow"]}

nastran.input.Design_Constraint = {"stress1": designConstraint1,
                                   "stress2": designConstraint2,
                                   "stress3": designConstraint3}
```

This completes the unique parts of the design inputs required for a Nastran optimization problem.

0.27.5 Composite Wing Example

This example introduces the use of composite materials. Initially a composite wing frequency analysis is completed. This example will grow to introduce design optimization with composites, including design variable linking.

Figure 2 Composite Wing Example

The CSM script generates Bodies which are designed to be used by specific AIMs. The AIMs that the Body is designed for is communicated to the CAPS framework via the “capsAIM” string attribute. This is a semicolon-separated string with the list of AIM names. Thus, the CSM author can give a clear indication to which AIMs should use the Body. In this example, the list contains the structural finite element analysis tools that can analyze the body:

```
attribute capsAIM $nastranAIM;astrosAIM;mystranAIM;egadsTessAIM
```

The parameter being set in this case is a definition for a coordinate system in Engineering Sketch Pad. The documentation for `csystem` inputs is as follows.

```
CSYSTEM   $csysName csysList
        use:    attach a Csystem to Body on top of stack
        pops:   any
        pushes: any
        notes:  Sketch may not be open
                if     csysList contains 9 entries:
                   {x0, y0, z0, dx1, dy1, dz1, dx2, dy2, dz2}
                   origin is at (x0,y0,q0)
                   dirn1  is in (dx1,dy1,dz1) direction
                   dirn2  is part of (dx2,dy2,dz2) that is orthog. to dirn1
                elseif csysList contains 5 entries and first is positive
                   {+iface, ubar0, vbar0, du2, dv2}
                   origin is at normalized (ubar0,vbar0) in iface
                   dirn1  is normal to Face
                   dirn2  is in (du2,dv2) direction
                elseif csysList contains 5 entries and first is negative
                   {-iedge, tbar, dx2, dy2, dz2}
                   origin is at normalized (tbar) in iedge
                   dirn1  is tangent to Edge
                   dirn2  is part of (dx2,dy2,dz2) that is orthog. to dirn1
                elseif csysList contains 7 entries
                   {inode, dx1, dy1, dz1, dx2, dy2, dz2}
                   origin is at Node inode
                   dirn1  is in (dx1,dy1,dz1) direction
                   dirn2  is part of (dx1,dy2,dz2) that is orthog. to dirn1
                else
                   error
                semicolon-sep lists can instead refer to
                   multi-valued Parameter
                dirn3 is formed by (dirn1)-cross-(dirn2)
                does not create a Branch
```

In the `compositesys` parameter defined below 9 entries are given. Based on the documentation above this indicates the following.

Origin	0.0	5.5	0.↩ 0
Vector along x-Axis	79.3685	-0.65432	0.↩ 0
Vector along y-Axis	0.65432	79.3685	0.↩ 0

It should be noted that the vector along the y-axis may not be input perfectly perpendicular to the vector along the x-Axis. In this case ESP takes the projection of the input vector that is in the plane defined by both input vectors and perpendicular to the x-Axis. This is the case for all csystem input options defined above.

```
dimension compositesys   9   1   0
set       compositesys 0;5.5;0;79.3685;-0.65432;0;0.65432;79.3685;0
```

The geometry definition was generated by the ESP sketcher. Users are referred to ESP tutorials for information on how to create a sketch. The result is copy and pasted into the `*.csm` file snippet shown below.

```
skbeg     0   0   0   1
   skvar     xy
      -0.024750;0.051384;4.841311;-0.024750;-3.895337;0.000000;82.067045;-3.895337;0.000000;77.594095;3.314007;0.000000;4.132
   skcon     X   1   -1  0
   skcon     Y   1   -1  0
   skcon     V   1   2  0
   skcon     L   3   4  8.5
   skcon     R   7   1  5.5
   skcon     H   2   3  0
   skcon     L   1   2  3.91
   skcon     L   6   7  3
   skcon     L   5   6  6.5
   skcon     L   4   5  74.2
   skcon     L   2   3  82.1
   skcon     P   1   -1  0
   skcon     A   5   -1  10
   skcon     A   6   -1  110
   skcon     A   4   -1  50
   linseg    ::x[2]   ::y[2]   0
   linseg    ::x[3]   ::y[3]   0
```

```
   linseg    ::x[4]   ::y[4]   0
   linseg    ::x[5]   ::y[5]   0
   linseg    ::x[6]   ::y[6]   0
   linseg    ::x[7]   ::y[7]   0
   arc       ::x[1]   ::y[1]   0   ::d[1]   xy
skend     0
```

The root edges are marked as `capsConstraint` locations. The overall surfaces is given a `capsGroup` and `capsLoad` attribute and a `csystem` definition is attached to it using the `compositesys` parameter previously discussed.

```
attribute capsGroup $wing
attribute capsLoad  $wing
attribute capsBound $wing
csystem   wing   compositesys
# Root edges
select    edge   6
attribute capsConstraint   $root
attribute .nPos  0

select    edge   7
attribute capsConstraint   $root
attribute .nPos  1

select    edge   1
attribute capsConstraint   $root
attribute .nPos  0

# Tip edge
select    edge   3
attribute .nPos  3

# Lower edge
select    edge   2
attribute .nPos  3

# Upper edges
select    edge   4
attribute .nPos  2

select    edge   5
attribute .nPos  0
```

This model was created in centimeters and is converted to inches.

```
scale 1/2.54
```

The following input defines a pyCAPS input that can be used along with the above $*$.csm input to create a nastran input. First the pyCAPS and os module needs to be imported.

```
# Import pyCAPS class file
import pyCAPS

# Import os module
import os
import shutil
import argparse
```

Once the modules have been loaded the pyCAPS.Problem needs to be initiated with the $*$.csm file. Though not shown in this example the user has access to the X, Y and Z despmtr inputs from this pyCAPS script.

```
# Load CSM file
geometryScript = os.path.join("..","csmData","compositeWing.csm")
capsProblem = pyCAPS.Problem(problemName=workDir,
                             capsFile=geometryScript,
                             outLevel=args.outLevel)
```

The Nastran AIM is then loaded with:

```
# Load nastran aim
nastran = capsProblem.analysis.create(aim = "nastranAIM",
                                      name = "nastran")
```

After the AIM is loaded some of the inputs to the AIM are defined. A full list of options can be found in the AIM Inputs section. In this case the `Proj_Name` is entered. The project name becomes the Nastran input file names. Two are create `projectName.bdf` and `projectName.dat`. The `*.bdf` file contains the grid and connectivity information. The data file contains the case control and other bulk data inputs required by Nastran. The input format is selected as `Small` and `Large` field format is used when the option is available. This is most likely in the GRID entries only. Additionally, the analysis type selected is `Static`. The maximum points that can be placed along an edge is set to be 40.

```
nastran.input["Mesh"].link(egads.output["Surface_Mesh"])

nastran.input.Proj_Name        = "nastran_CompositeWing"
nastran.input.File_Format      = "Small"
nastran.input.Mesh_File_Format = "Large"
nastran.input.Analysis_Type    = "Modal"
```

In this example two materials are defined. This demonstrates how simple it is to change materials in a model. Both an aluminum and Graphite_expoxy material are defined in the `Material` AIM Inputs.

```
Aluminum  = {"youngModulus" : 10.5E6 ,
             "poissonRatio" : 0.3,
             "density"      : 0.1/386,
             "shearModulus" : 4.04E6}

Graphite_epoxy = {"materialType"        : "Orthotropic",
                  "youngModulus"        : 20.8E6 ,
                  "youngModulusLateral" : 1.54E6,
                  "poissonRatio"        : 0.327,
                  "shearModulus"        : 0.80E6,
                  "density"             : 0.059/386,
                  "tensionAllow"        : 11.2e-3,
                  "tensionAllowLateral" : 4.7e-3,
                  "compressAllow"       : 11.2e-3,
                  "compressAllowLateral": 4.7e-3,
                  "shearAllow"          : 19.0e-3,
                  "allowType"           : 1}

nastran.input.Material = {"Aluminum": Aluminum,
                          "Graphite_epoxy": Graphite_epoxy}
```

Again property information is defined for both an aluminum and composite stack version of the model. However, only the composite entry is defined in the `Property` AIM Inputs.

The `composite` definition brings together the materials in each layer of the stack, their thicknesses and orientations. For the sequence defined below the order of the sequence is given in the table below. The full sequence is given to point out that they symmetry condition is applied to the right side of the `compositeOrientation` input.

0	0	0	0	-45	45	-45	45	45	-45	45	-45	0	0	0	0

Finally, the property is assigned to the regions with the `capsGroup $wing` attribute.

```
aluminum  = {"propertyType"         : "Shell",
             "material"             : "Aluminum",
             "bendingInertiaRatio"  : 1.0, # Default - not necesssary
             "shearMembraneRatio"   : 0, # Turn of shear - no materialShear
             "membraneThickness"    : 0.125 }

composite  = {"propertyType"           : "Composite",
              "shearBondAllowable"       : 1.0e6,
              "bendingInertiaRatio"    : 1.0, # Default - not necesssary
              "shearMembraneRatio"     : 0, # Turn off shear - no materialShear
              "compositeMaterial"      : ["Graphite_epoxy"]*8,
              "compositeThickness"     : [0.00525]*8,
              "compositeOrientation"   : [0, 0, 0, 0, -45, 45, -45, 45],
              "symmetricLaminate"      : True,
              "compositeFailureTheory" : "STRN" }

#nastran.input.Property" = {"wing": aluminum}
nastran.input.Property = {"wing": composite}
```

In this example the root edges are constrained in all degrees of freedom. This constraint references the `caps↩Constraint $root` input defined in the *.csm file.

```
constraint = {"groupName" : "root",
              "dofConstraint" : 123456}

nastran.input.Constraint = {"root": constraint}
```

As previously shown in Modal Analysis Example Case information to define an eigen value problem is entered and the `Analysis` AIM Input is defined.

```
eigen = { "extractionMethod"     : "MGIV",#"Lanczos",
          "frequencyRange"       : [0, 10000],
          "numEstEigenvalue"     : 1,
          "numDesiredEigenvalue" : 10,
```

```
          "eigenNormalization"   : "MASS"}

nastran.input.Analysis = {"EigenAnalysis": eigen}
```

Finally, preAnalysis is executed to generate all the required Nastran inputs.

```
nastran.preAnalysis()
```

Nastran is executed with a simple system call.

```
print ("\n\nRunning Nastran......" )

if args.noAnalysis == False:
    nastran.system("nastran old=no notify=no batch=no scr=yes sdirectory=./ " + nastran.input.Proj_Name +
      ".dat"); # Run Nastran via system call
else:
    nastran.system("test_bdf -e 0 " + nastran.input.Proj_Name +  ".dat"); # Run pyNastran test_bdf via
      system call
    # Copy old results if no analysis available
    shutil.copy2(os.path.join("..","analysisData","nastran",projectName+".f06"),
                 os.path.join(nastran.analysisDir,nastran.input.Proj_Name+".f06"))
    shutil.copy2(os.path.join("..","analysisData","nastran",projectName+".op2"),
                 os.path.join(nastran.analysisDir,nastran.input.Proj_Name+".op2"))

print ("Done running Nastran!")
```

A post analysis command is entered allowing the user to access output data, if desired, from the application.

```
# Run AIM post-analysis
nastran.postAnalysis()
```

0.27.6 Composite Wing Optimization Example

This section removes the frequency analysis and adds a pressure load to the previously introduced composite wing example case. Then An optimization problem allowing the thickness of each ply layer to change is performed.

Bibliography

[1] Michael Reymond and Mark Miller. *MSC NASTRAN Quick Reference Guide Version 68*, 1996. 1

Index